HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

# TIỂU LUẬN MÔN HỌC
# KIẾN TRÚC VÀ THIẾT KẾ PHẦN MỀM


**Giảng viên hướng dẫn** : **Trần Đình Quế**
**Họ và tên sinh viên** : **Nguyễn Minh Tuân**
**Mã sinh viên** : **B20DCCN036**
**Lớp** : **D20CNPM5**
**Nhóm** : **13**


***Hà Nội – 2024***

# MỤC LỤC

# MỞ ĐẦU

Ngoài Mở đầu và Kết luận, báo cáo được cấu trúc thành 3 Chương dưới đây.

# Chương 1

# Kiến trúc monolithic và microservice

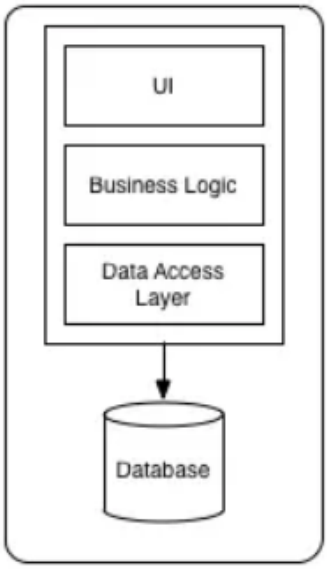
## 1.1. Giới thiệu

Software architecture là tổ chức hệ thống bao gồm rất nhiều các thành phần như Web Server, cơ sở dữ liệu, bộ nhớ và các lớp layer thực hiện việc giao tiếp. Chúng liên kết với nhau hoặc với một môi trường nhất định. Mục tiêu cuối cùng của thiết kế hệ thống (system architecture) là giải quyết vấn đề của doanh nghiệp.

Ở thời điểm hiện tại, có 2 mô hình pattern của software architecture đang được phổ biến là:

- Monolith architecture
- Mircoservice architecture

## 1.2. Kiến trúc monolithic




Monolithic Architecture

Monolith có xu hướng phù hợp với những dự án có quy mô nhỏ. Với việc áp dụng mô hình monolith, những lợi ích đem lại có thể kể đến là:

- Quá trình development đơn giản và trực tiếp, centralized managenment và những bước phát triển cơ bản thì sẽ không được lặp lại.

- Effort dành cho việc development được giảm thiểu: tất cả mọi quá trình development đều nằm trên 1 project. Development flow đơn giản chỉ là submit changes, review, merge code và continue.

Tuy nhiên hạn chế mà mô hình này đem lại cũng khá lớn :

- Khó khăn trong việc bảo trì: vấn đề về coupling code, các khối code dính chặt lại với nhau, vấn đề cho member mới sẽ khó để biết nên bắt đầu từ đâu trong 1 khối lớn
- Quá trình development sẽ mất đi tính linh hoạt: thời gian để build feature sẽ bị dài lên, bị block lẫn nhau. Bất kì một sự thay đổi dù nhỏ nào cũng cần build lại toàn bộ dự án => tốn khá nhiều thời gian
- Tính ổn định không cao. Bất kì một lỗi nào có thể khiến toàn bộ application bị crash.
- Tính scalibility khó được đáp ứng trong trường hợp phải đáp ứng một lượng truy cập lớn từ phía yêu cầu của business

### 1.3. Kiến trúc microservice

Ngoài mô hình monolithic kể trên, hiện nay có 1 architecture khác đang nhận được nhiều sự quan tâm, đó là microservice. Microservice đề cập đến quá trình phát triển độc lập, tương đối nhỏ theo hướng chia hệ thống ra thành các services. Mỗi service này đều có một logic riêng, một trách nhiệm riêng và có thể được deploy riêng biệt. Khái niệm mircoservice đồng thời đề cập đến xu hướng tách biệt architecture ra thành các loose coupling service, tức là các service này sẽ có một mối liên hệ lỏng lẻo với nhau và mối service sẽ được nằm trong 1 context nhất định.

So sánh với microservice và SOA (service-oriented architecture), những điểm khác biệt của mô hình microservice là componentization (thành phần hóa), loose coupling (khớp nối lỏng lẻo), autonomy ( tính tự quản lí) và decentralization (phân cấp), được phản ánh cụ thể qua những khía cạnh sau:

- Tập hợp một nhóm nhỏ các service: mức độ chi tiết của một service là nhỏ và mỗi service này sẽ chịu một trách nhiệm cụ thể (single responsiblity) và chỉ tập trung vào nhiệm vụ đó. Ví dụ: storage service sẽ chịu riêng trách nhiệm về lưu trữ
- Việc phát triển và mở rộng một service là hoàn toàn độc lập. Điều này mang lại tính linh hoạt cho hệ thống . Quá trình deliver feature, release

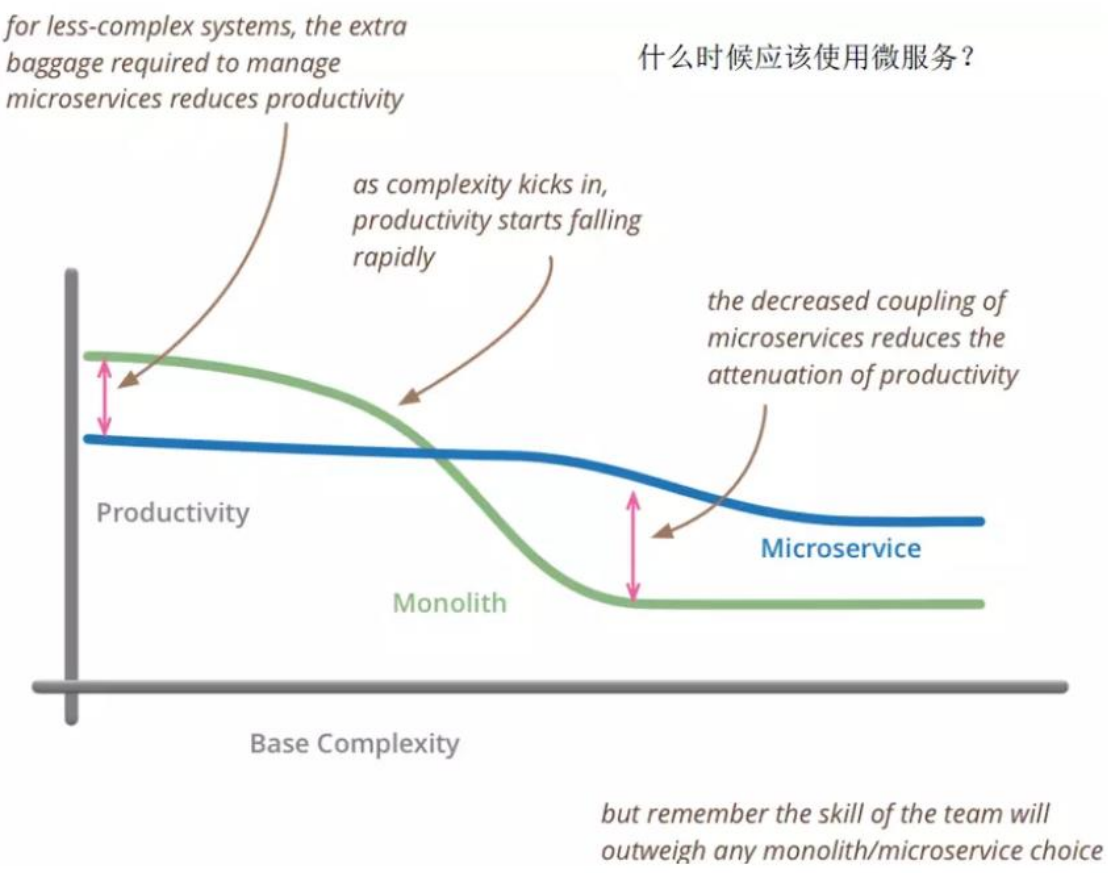
version sẽ dễ dàng và nhanh chóng. Hơn nữa sẽ không còn tình trạng bị block như ở mô hình monolith

- Giảm tải được các mối quan ngại về công nghệ sử dụng. Chọn một công nghệ phù hợp với vấn đề của doanh nghiệp có thể được giải quyết dễ dàng. Các service giap tiếp với nhau thông qua API, do vậy mỗi service có thể dùng một ngôn ngữ riêng biệt. Serivce A dùng Java, Service B dùng Javascript, it's ok !!!!
- Đối với team, microservice đem lại tính độc lập và tự quản lí cho team. Một team sẽ có trách nhiệm toàn bộ với life-cycle của một hay nhiều service. Họ làm việc trong việc context biệt lập, có thể tự quản lí các quyết định của mình.

Chúng ta có thể thấy rõ toàn bộ ý tưởng của mô hình microservice rất giống cách mà chúng ta chia nhỏ thông tin và kiến thức. Bằng việc tách rời, chia nhỏ và quản lí chúng ta có thể giảm tải sự phức tạp của hệ thống, làm cho việc quản lí trở nên nhanh chóng và dễ dàng, phản ánh sự thay đổi chính xác.

**Tại sao nên dùng microservice?**

Ở thế kỷ trước, một số lightweight development methods như eXtreme Programming (XP) hay Scrum nổi lên; Đến năm 2001, tuyên ngôn Agile ra đời và một số phương pháp quản lý mới như Lean hay Kanban. Nếu những phương pháp quản lý trên được coi là giải pháp cho việc quản lý tiến độ phát triển phần mềm và việc thực hiện sớm nhất có thể khi có sự thay đổi thì microservice architecture là hướng tiếp cận được nói đến trong công nghệ phần mềm và ở tầng kiến trúc (architecture level). Dưới đây là một biểu đồ so sánh giữa monolith và microservice:




**Nhiều doanh nghiệp mong muốn có tính linh hoạt của microservices nhưng cảm thấy rủi ro cao trong quá trình chuyển đổi?**
Với công nghệ tiên tiến cùng đội ngũ nhân sự chuyên môn cao, nhiều năm kinh nghiệm trong tư vấn, thiết kế và triển khai hệ thống của Sunteco, doanh nghiệp có thể chuyển đổi kiến trúc microservices sang monolithic. Các nhà phát triển có thể nâng cấp, chuyển đổi hệ thống một cách nhanh chóng mà không cần sử dụng nhiều phần mềm trung gian phức tạp và không ảnh hưởng đến hoạt động của hệ thống hiện tại. Sự phân chia quy trình và chức năng của hệ thống này cho phép module hóa và mang lại tính linh hoạt cao hơn.

Các thuộc tính của mô hình microservice:

- **Autonomous** (tính tự trị): 1 service sẽ là 1 đơn vị chức năng, cung cấp API để thực hiện việc trao đổi, giao tiếp với các service khác

- **Isolated** (tính biệt lập): 1 serivce sẽ là 1 đơn vị triển khai. Nó có thể được chỉnh sửa, test và deployed như 1 đơn vị mà không ảnh hưởng đến những khía cạnh khác.
- **Elastic**: 1 service là phi trạng thái (stateless) vì vậy nó có thể scale tùy ý khi cần thiết.
- **Resilient**: 1 microservice sẽ được thiết kế để chấp nhận các lỗi, các rủi ro có thể xảy ra, các lỗi này là các lỗi có thể chấp nhận được
- **Responsive**: Respond cho các request trong khoảng thời gian hợp lý.
- **Intelligent**: Tính thông minh ở đây tức là muốn nhắc đến việc hệ thống có thể tìm thấy các endpoint của các microservice đã được đăng kí.
- **Message Oriented**: Mô hình micro-service hoạt động dựa trên giao thức HTTP hoặc message bus để tạo nên sự giao tiếp giữa các service. Điều này đảm bảo tính loose coupling, tính biệt lập và có thể cung cấp lỗi dưới dạng message
- **Programmable**: Cung cấp API's cho phép truy cập bởi developer và administrator.
- **Composable**: Bao gồm nhiều microservices.
- **Automated**: Lifecycle của Microservice được quản lý thông qua automation bao gồm development, build, test, staging, production và distribution.)

**Ưu điểm của microservice:**

- Mỗi microservice sẽ được chia nhỏ để tập trung vào một business function cụ thể hoặc business requirement.
- Microservices có thể phát triển độc lập bởi một team nhỏ có thể chỉ từ 2 đến 5 developers.
- Microservice đem lại tính loose-coupling và context riêng cho mỗi service, sẽ dễ dàng trong quá trình development cũng như deploy một cách độc lập..
- Microservices có thể phát triển với nhiều ngôn ngữ khác nhau.
- Quá trình phát triển một service sẽ trở nên dễ dàng và linh động thông qua việc sử dụng CI/CD như Travis, Jenskin, Circle CI ....
- 1 new member có thể dễ dàng và nhanh chóng đóng góp cho dự án

- 1 serive trong mô hình micro serivce là tương đối nhỏ, dễ hiểu và được quản lí bởi các thành viên của 1 team nhỏ. Do đó, họ sẽ dễ dàng tập trung vào công việc, nâng cao được hiệu năng.
- Microservices cho phép tận dụng việc áp dụng những công nghệ mới vào dự án.
- Microservices chỉ gồm business logic code và không bao gồm HTML, CSS.
- Việc deploy sẽ mất ít effort cho việc configuraton.
- Dễ dàng tích hợp 3rd-party.
- Mỗi service có dung lượng lưu trữ riêng và có thể có cơ sở dữ liệu riêng.

**Nhược điểm:**

- Microservice architecture có thể dẫn tối việc sử dụng quá nhiều operations.
- Cần thêm kiến thức về DevOps
- Effort của team có thể sẽ phải x2
- Distributed systems (hệ thống phân tán) phức tạp và khó quản lý
- Số lượng service càng lớn thì vấn đề về management complexity cũng tăng theo.

## 1.4. Phân rã Hệ thống phần mềm thành các service

- Lợi ích của việc phân rã
    - Tính linh hoạt và độc lập: Mỗi service có thể được phát triển, triển khai, vận hành, và mở rộng một cách độc lập.
    - Tối ưu hóa tài nguyên: Các service có thể được mở rộng một cách độc lập tùy vào nhu cầu sử dụng, giúp tối ưu hóa việc sử dụng tài nguyên hệ thống.
    - Khả năng chịu lỗi: Sự cố ở một service không nhất thiết ảnh hưởng đến toàn bộ hệ thống.
    - Tính tái sử dụng: Các service được thiết kế để có thể tái sử dụng trong các bối cảnh khác nhau.

- Chiến lược phân rã

    - Dựa trên chức năng nghiệp vụ: Phân chia hệ thống dựa vào các chức năng nghiệp vụ khác nhau. Mỗi chức năng nghiệp vụ trở thành một service riêng biệt.
    - Phân tầng dịch vụ: Chia nhỏ hệ thống thành các tầng khác nhau như tầng trình diễn, tầng kinh doanh, và tầng dữ liệu, với mỗi tầng có thể có một hoặc nhiều service.
    - Dựa trên các tiêu chí đặc thù: Như tần suất sử dụng, yêu cầu đối với tài nguyên hệ thống, hoặc độ nhạy cảm của dữ liệu.

- Công cụ và các mô hình thiết kế
    - API và cổng giao tiếp: Các service giao tiếp với nhau thông qua API (Application Programming Interface) định nghĩa rõ ràng.
    - Message Queue và Event Stream: Sử dụng hàng đợi tin nhắn (như RabbitMQ, Kafka) để xử lý giao tiếp giữa các service một cách bất đồng bộ.
    - Containers và Orchestration: Sử dụng các công cụ như Docker và Kubernetes để đơn giản hóa việc triển khai và quản lý các service.

- Khó khăn
    - Phức tạp trong quản lý: Việc vận hành một hệ thống được phân chia thành nhiều service có thể trở nên phức tạp hơn.
    - Hiệu suất giao tiếp: Việc phụ thuộc quá nhiều vào giao tiếp mạng giữa các service có thể ảnh hưởng đến hiệu suất.
    - Tính nhất quán dữ liệu: Đảm bảo tính nhất quán dữ liệu giữa các service là một thách thức đáng kể.

## 1.5. Kết luận

Thông qua những phân tích đặc điểm và trường hợp cụ thể sử dụng của từng mô hình, bài viết có thể giúp doanh nghiệp có cái nhìn tổng quan hơn về ứng dụng của nó. Nếu quý khách hàng đang gặp phải vấn đề trong lựa chọn giữa kiến trúc Microservices vs Monolithic và cần sự tư vấn chi tiết hơn, liên hệ với chúng tôi để nhận được sự hỗ trợ và tìm ra giải pháp phù hợp nhất cho nhu cầu của doanh nghiệp.

# Chương 2

# Kiến trúc Hệ thống ecomSys dựa trên Django

## 2.1. Mô tả Hệ ecomSys

## 2.2. Kỹ thuật và công nghệ Django

## 2.3. Phân rã Hệ ecomSys dựa trên Django

## 2.4. Kết luận

# Chương 3

# Thiết kế và xây dựng hệ ecomSys

## 3.1. Giới thiệu

Hệ ecomSys được thiết kế theo kiến trúc microservice, với từng service sẽ sử dụng database riêng biệt được lưu trữ trên cloud, ví dụ như service user dùng PostgreSQL; service mobile, clothes và book sử dụng MongoDB; service cart, payment, shipment, order sử dụng MySQL. Các service sẽ cung cấp và giao tiếp với nhau qua API để lấy dữ liệu liên quan.

Dưới đây là mô tả chức năng trong hệ ecomSys

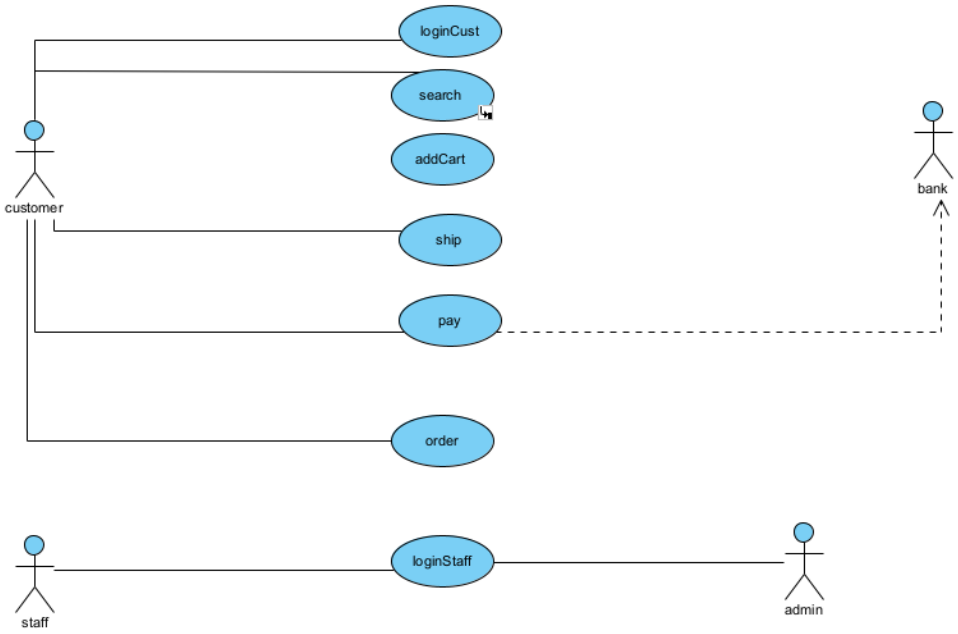
| Actor | Functionalities | Detailed Description |
|---|---|---|
| Customer | Authenticate | - Đăng ký: khách hàng đăng ký tài khoản sử dụng hệ thống ecommerce qua thông tin cá nhân như họ tên, số điện thoại, email, …<br>- Đăng nhập: khách hàng đăng nhập sử dụng hệ thống để có thể sử dụng các chức năng như mua hàng, thanh toán, …<br>- Đăng xuất: khách hàng đăng xuất khỏi hệ thống |
| | Search | - By Browser: người sử dụng có thê browser trên web và chọn mặt hàng mình muốn để thêm vào **cart.** NSD có thể xem chi tiết mặt hàng.<br>- By keyword: người dùng có thể tìm sản phẩm bằng cách nhập keyword thuộc về sản phẩm vào thanh tìm kiếm<br>- By image: người dùng có thể tìm sản phẩm bằng cách upload ảnh sản phẩm muốn tìm lên<br>- By voice: nsd nói tên (mã,…) mặt hàng mình muốn |

| | | |
|---|---|---|
| | Add to cart | - Người dùng có thể thêm sản phẩm muốn mua vào giỏ hàng, giỏ hàng sẽ lưu các sản phẩm đó mỗi khi người dùng quay lại sử dụng hệ thống |
| | Order & checkout | - Sau khi thêm sản phẩm vào giỏ hàng, người dùng nhấn thanh toán và khi đó hệ thống sẽ tạo 1 đơn hàng, người dùng có thể xem lại những mặt hàng có trong giỏ hàng và cung cấp thông tin cần thiết như họ tên, địa chỉ giao hàng, … |
| | Pay | - Người dùng có thể chọn các phương thức thanh toán khác nhau (thẻ ngân hàng, thanh toán tiền mặt, …) cho mặt hàng mình muốn mua và sau đó thanh toán |
| Super Admin (đóng vai trò là manager) | Authenticate | - Hệ thống cung cấp sẵn 1 tài khoản Super Admin để quản lý hệ thống như quản lý người dùng, quản lý admin con (nhân viên hệ thống) |
| | Customer management | - Xem thông tin khách hàng đã đăng ký vào hệ thống |
| | Admin management | - Super admin quản lý tài khoản các admin con (nhân viên hệ thống ecommerce), có thể tạo mới tài khoản, cập nhật thông tin tài khoản và xóa tài khoản |
| Admin | Authenticate | - Admin con được hiểu là nhân viên hệ thống, được Super Admin tạo tài khoản đăng nhập vào hệ thống để quản lý sản phẩm (book, mobile, clothes), quản lý đơn hàng, … |
| | Book management | - Admin có thể thêm mới, cập nhật và xóa mặt hàng sách |
| | Mobile management | - Admin có thể thêm mới, cập nhật và xóa mặt hàng điện thoại |
| | Clothes management | - Admin có thể thêm mới, cập nhật và xóa mặt hàng thời trang |
| | Order management | - Admin có thể xem thông tin của tất cả đơn hàng, chuyển trạng thái đơn hàng và xóa đơn hàng |
| Shipment | Ship | - Sau khi giai đoạn thanh toán xử lý thành công, quá trình chuẩn bị và giao hàng cho khách bắt đầu, gồm đóng gói, phân loại và vận chuyển hàng đến địa chỉ mà khách đã cung cấp ở quá trình checkout |

## 3.2. Các biểu đồ đặc tả hệ ecomSys

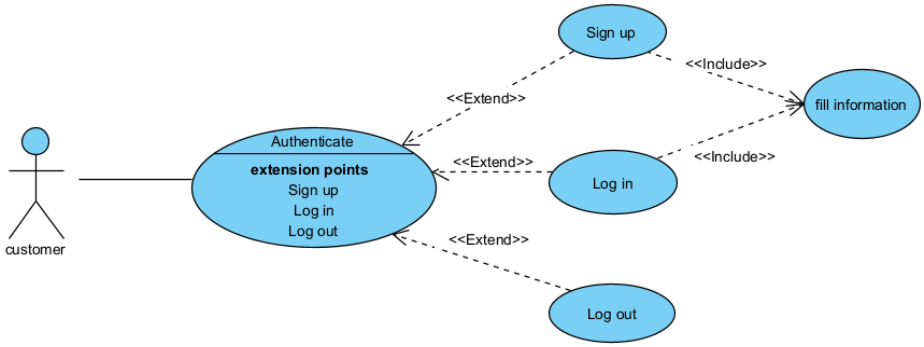
### 3.2.1. Biểu đồ use case

#### 3.2.1.1. General use case diagram

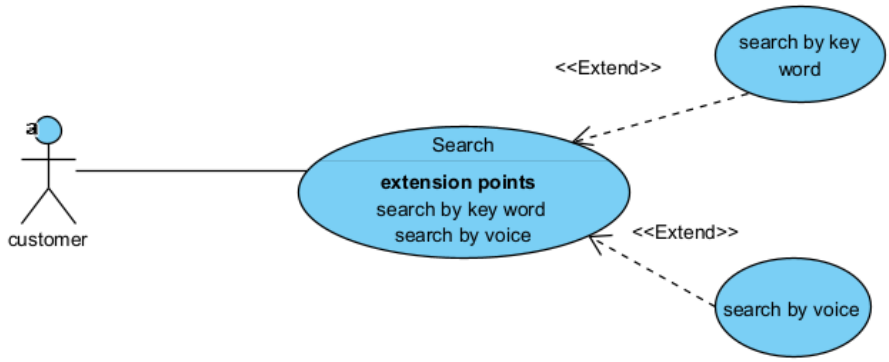



### 3.2.1.2. Actor customer

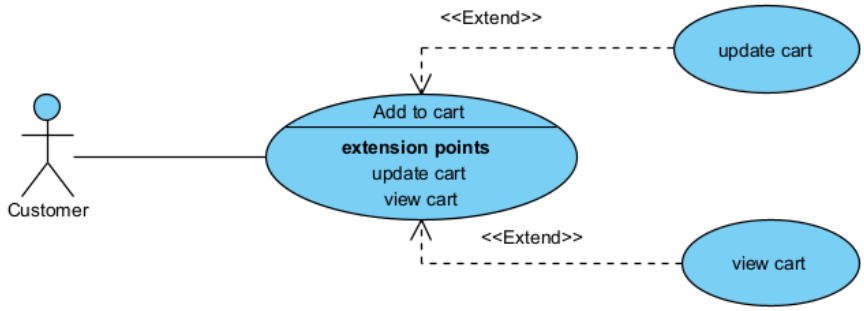
- Detailed Use case for authentication



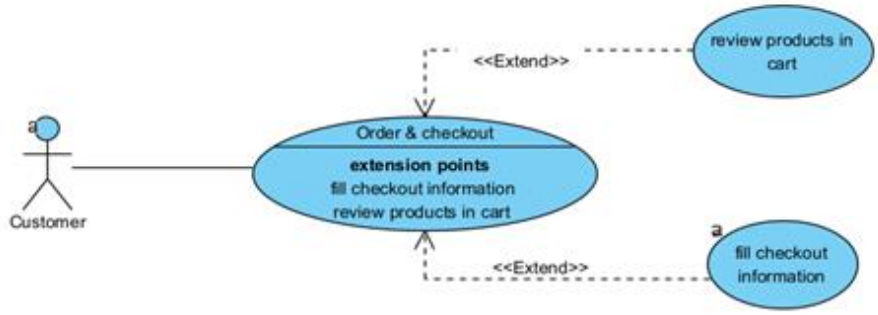

- Detailed Use case for search




- Detailed Use case for Add to cart

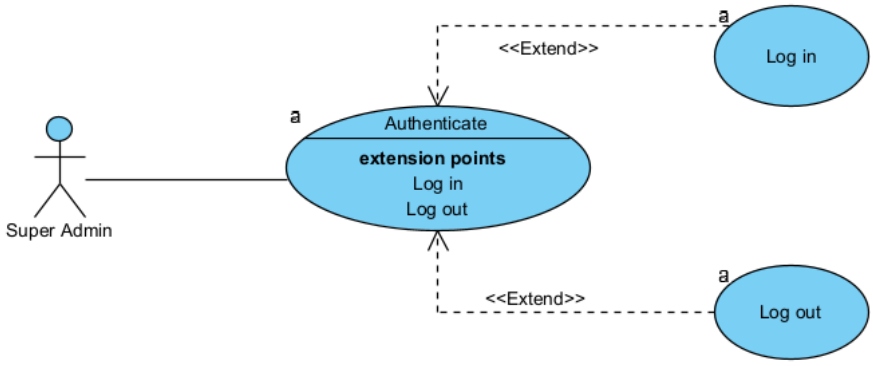



- Detailed Use case for order & checkout

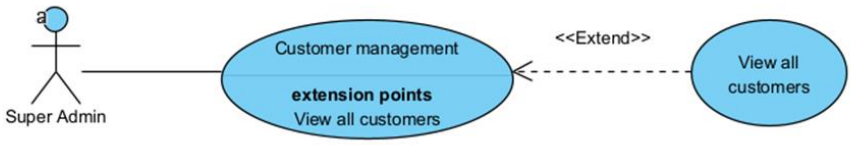



### 3.2.1.3. Actor Super Admin

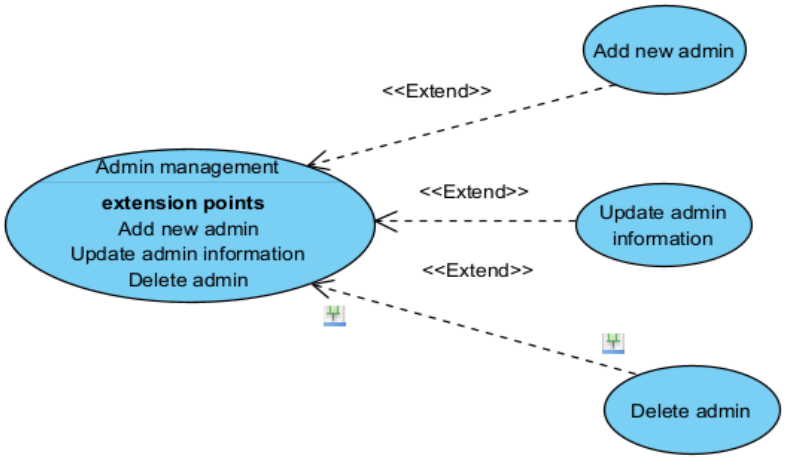
- **Detailed Use case for authentication**




- **Detailed Use case for customer management**

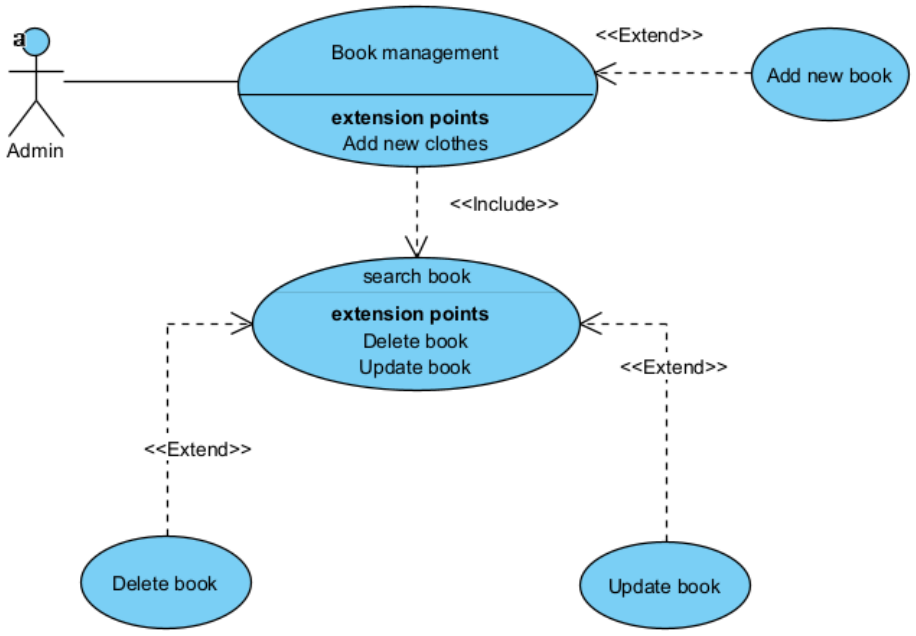



- **Detailed Use case for Admin management**

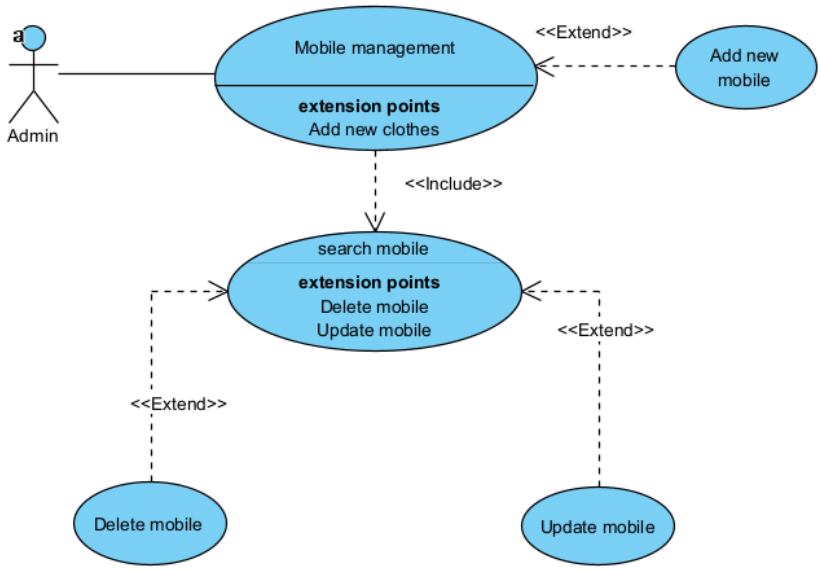



### 3.2.1.4. Actor Admin

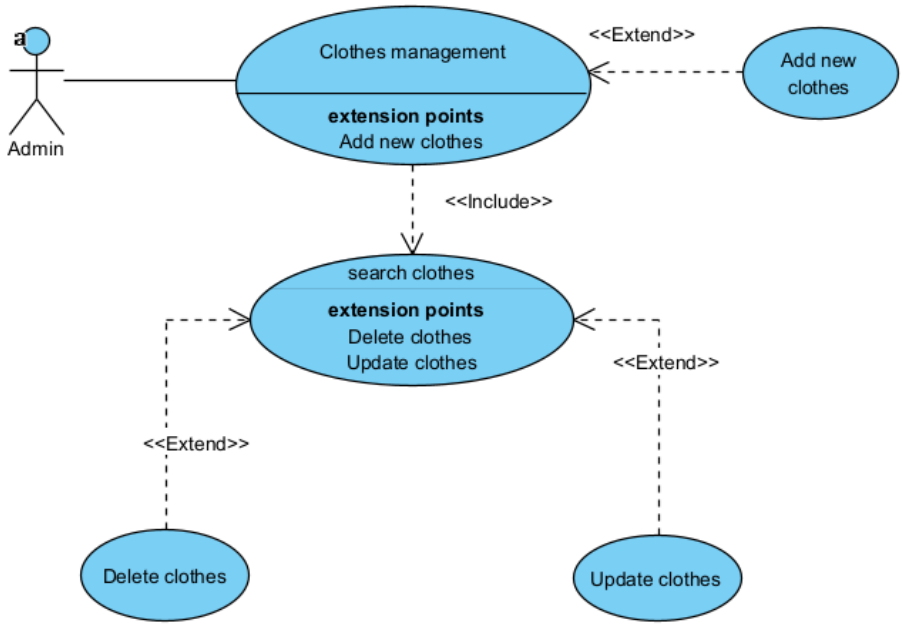
- **Detailed Use case for Book management**



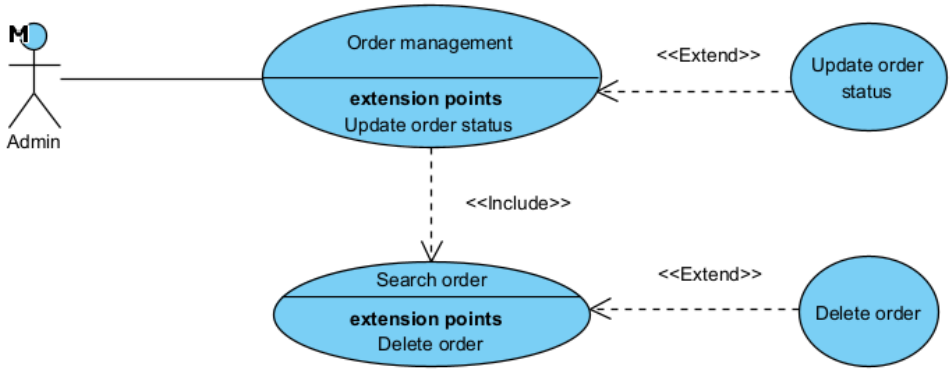

- **Detailed Use case for Mobile management**



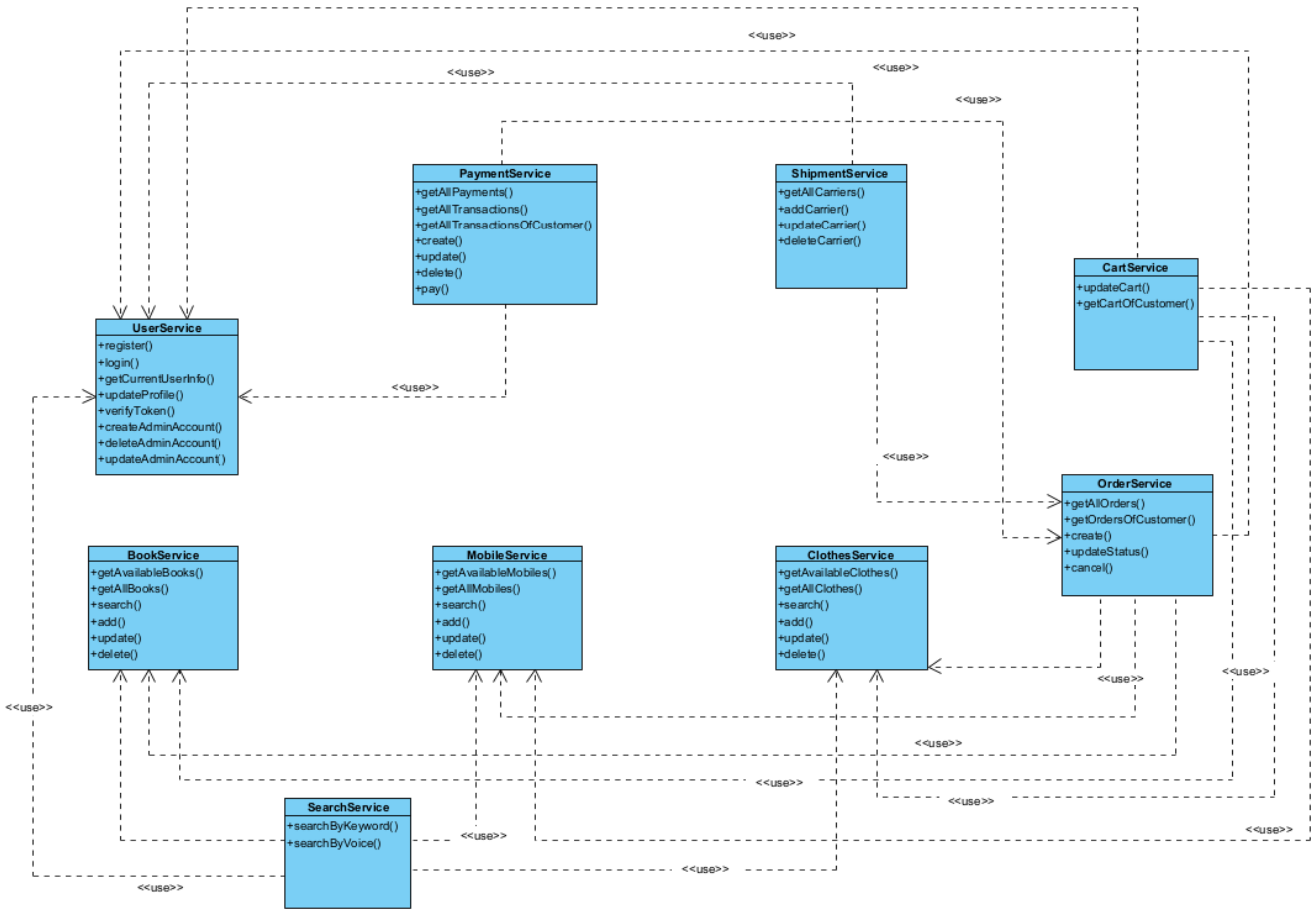

- **Detailed Use case for Clothes management**




- **Detailed Use case for order management**




## Decompose a software systems into module/services




### 3.2.2. Biểu đồ hoạt động/swimlane

### 3.2.3. Biểu đồ lớp phân tích cho từng service

### 3.2.4. Mô hình Data- kiểu/công nghệ dữ liệu

### 3.2.5. Biểu đồ lớp thiết kế cho từng service

## 3.3. Xây dựng hệ ecomSys

### 3.3.1. Module (service/project) user

### 3.3.2. Module cart

### 3.3.3. Module search

### 3.3.4. Module product

### 3.3.5. Module order

### 3.3.6. Module payment

### 3.3.7. Module shipment

## 3.4. Thiết kế giao diện

## 3.5. Thiết kế database

## 3.6.

## Kết luận